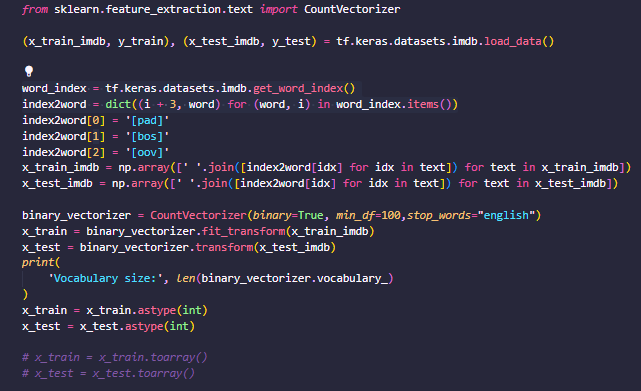
|  |  |  |  |
| --- | --- | --- | --- |
|  | Τεχνητή Νοημοσύνη 2η Εργασία | |  |
|  | |
|  | |
|  | Ταξινόμηση κριτικών IMDB | |  |
| * Σοφία-Ζωή Σωτηρίου : 3210192 * Ευάγγελος Λευτάκης : 3200093 * Ρέα Σκλήκα : 3210181 |  |  |
|  | |  |

Αρχικοποίηση Δεδομένων

Αφού λάβουμε το imdb dataset μαζί το λεξικό word index του μέσω του Keras, προσθέτουμε στο λεξικό τις λέξεις [pad], [bos], [oov] που είναι tokens με ειδική σημασία για την αναπαράσταση κειμένου. Έπειτα με την χρήση του countVectorizer φιλτράρουμε τις λέξεις και κρατάμε μόνο αυτές που εμφανίζονται τουλάχιστον 100 φορές (min\_df) και παράλληλα μετατρέπουμε τις προτάσεις σε δυαδικές αναπαραστάσεις από 0 για απουσία και 1 για εμφάνιση μιας λέξης σε κάθε πρόταση. Αλλάζουμε τον τύπο τον δεδομένων από float64 που είναι το default σε integer για βελτιωμένη απόδοση. Τέλος, μετατρέπουμε τα δεδομένα μάθησης και ελέγχου σε dense arrays με την χρήση της toarray() καθώς by default λαμβάνουμε ένα sparse array από τον count vectorizer η οποία μορφή δεν είναι συμβατή με πολλές από τις διαδικασίες που απαιτούνται για τις διάφορες μεθόδους τις εργασίας.



1ο Μέρος

Για να κάνουμε την κατηγοριοποίηση των κριτικών από το dataset μας στο πρώτο κομμάτι της εργασίας αναπτύξαμε APIs για τους αλγορίθμους μηχανικής μάθησης Naive Bayes, Logisitic Regression και AdaBoost και να αξιολογήσουμε τις επιδόσεις τους.

Logistic regression:

Ο αλγόριθμος που υλοποιήσαμε είναι λογιστικής παλινδρόμησης με L2 κανονικοποίηση και κανόνα ανανέωσης βαρών: 

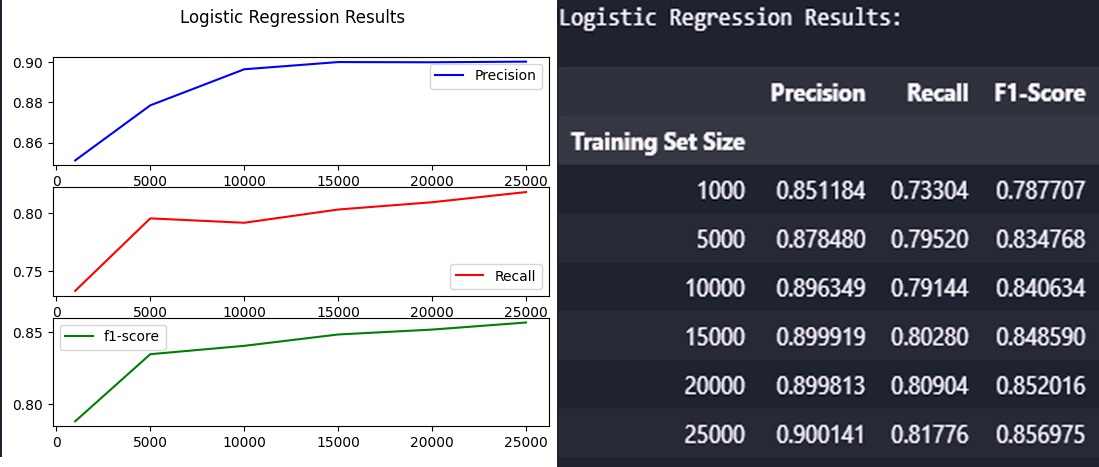
Η διεπαφή αποτελείται από τις παρακάτω μεθόδους:

* \_\_init\_\_(*epochs* ,*learning\_rate* ,*threshold* ,*regularization\_factor*): Παίρνει ως ορίσματα τις διάφορες υπερπαραμέτρους όπως αριθμός επαναλήψεων, ρυθμός μάθησης παράγοντας κανονικοποίησης και threshold. Κατασκευάζει και επιστρέφει τον ταξινομητή με τα ορίσματα που προσδιορίσαμε.
* fit(x\_train\_input,y\_train\_input,learning\_curve): Παίρνει ως ορίσματα τα δεδομένα εκπαίδευσης και μια boolean τιμή σε περίπτωση που θέλουμε να παραχθεί καμπύλη μάθησης. Αρχικά διαιρούμε τα δεδομένα σε εκπαίδευσης και επικύρωσης 80-20 αντίστοιχα. Για κάθε εποχή τα παραδείγματα αναδιατάσσονται τυχαία προς αποφυγή overfitting και να μην δημιουργηθούν γνώσεις που έχουν να κάνουν με την διάταξη των παραδειγμάτων εκπαίδευσης. Για την ενημέρωση των βαρών χρησιμοποιούμε Stochastic Gradient Ascent χωρίς mini batches. Επιστρέφει διάφορες μετρικές που παράγονται από την evaluate() (εμφανίζεται παρακάτω
* predict(x\_test): Παίρνει ως όρισμα τα δεδομένα ελέγχου. Με τον τύπο τις σιγμοειδούς καμπύλης υπολογίζουμε την πιθανότητα κάθε παράδειγμα ελέγχου να είναι στην θετική κατηγορία. Έπειτα αν η πιθανότητα αυτή είναι μεγαλύτερη από το threshold θωρούμε ότι η κριτική είναι θετική. Επιστρέφει έναν πίνακα y\_predicted με μήκος ίσο με τα παραδείγματα ελέγχου ο οποίος αποτελείται από 0 και 1 για εκτιμόμενα θετική ή αρνητική κριτική.
* predict\_prob(x\_test): Το ίδιο με την predict απλά επιστρέφει τις εκτιμώμενες πιθανότητες για τα δεδομένα ελέγχου χωρίς να τα κατατάσσει.
* initialize\_weights(): Αρχικοποιούμε τα βάρη σε τυχαίους αριθμούς πολύ κοντά στο μηδέν.
* update\_weights(x\_train, y\_train): Χρησιμοποιώντας sga με τον τύπο ανανεώνουμε τα βάρη με τον τύπο ανανέωσης.
* evaluate(y\_test, y\_predicted): Συγκρίνουμε τα αποτελέσματα που εκτιμήθηκαν με τα πραγματικά και επιστρέφουμε ένα πίνακα με μετρικές όπως [precision,recall,f1]
* get\_params(): Υλοποιούμε για συμβατότητά με μεθόδους του sci-kit learn όπως learning\_curve()
* classification\_diagrams(x\_train, y\_train, x\_test, y\_test, train\_sizes): Παράγει τρία plots για την πορεία των precision, recall, f1 score συναρτήσει του μεγέθους των δεδομένων εκπαίδευσης.
* custom\_classification\_report(*x\_train*, *y\_train*,*x\_test*, *y\_test*, *train\_sizes*): Καλεί την classification\_diagrams() και την my\_learning\_curve() για να δώσει μια πλήρη εικόνα των αποτελεσμάτων μάθησης του ταξινομητή που π.
* Άλλες Βοηθητικές μέθοδοι όπως set\_threshold() για αλλαγή του threshold ενός ταξινομητή, sigmoid() για υπολογισμό του τύπου .

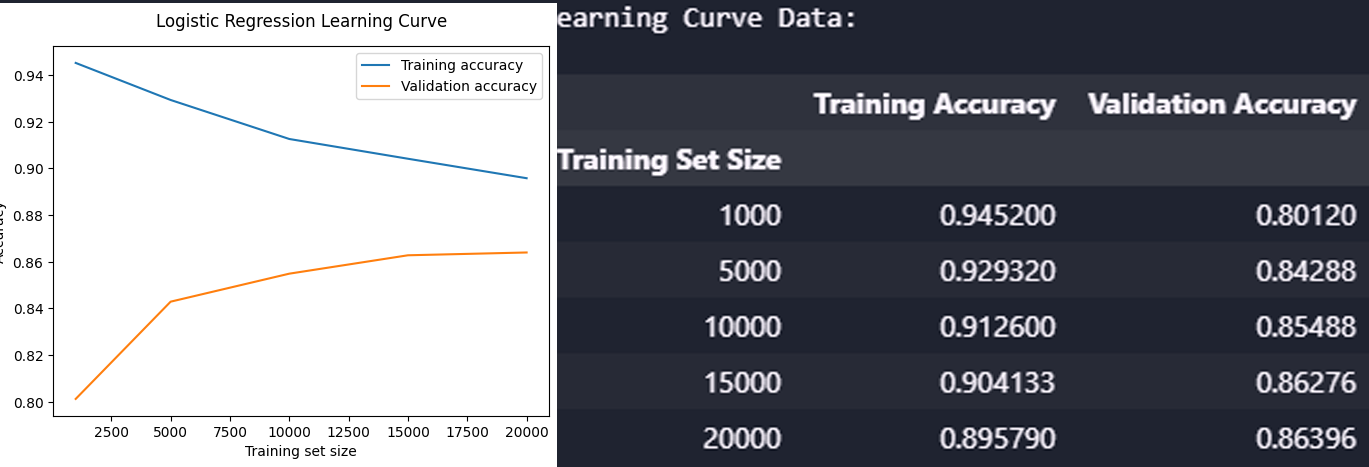
Αποτελέσματα:

Ο ταξινόμητής που χρησιμοποιήσαμε για τα δεδομένα μάθησης τελικά είχε τις ύπερ παραμέτρους:

* Threshold: του οποίου δώσαμε διάφορες τιμές και καταλήξαμε ότι κάτι κοντά στο (0.4,0.6) έχει καλά αποτελέσματα και μια καλή ισορροπία ανάμεσα σε precision και recall.
* Regularization Factor λ: βάζουμε χαμηλό παράγοντα κανονίκοποίησης (0.001) καθώς για το πρόβλημα μας που είναι sentiment analysis επωφελούμαστε από την καλή αφομοίωση των δεδομένων εκπαίδευσης και αποφεύγοντας το overfitting
* Learning Rate: Αφήνουμε την τιμή στο default που έχουμε ορίσει 0.001 που προσφέρει πιο αργό ρυθμό μάθησης αλλά περιορίζει το ενδεχόμενο να χάσουμε κατά πολύ τα βέλτιστα βαρύ και να ταλαντευόμαστε γύρω από την μέγιστη τιμή πιθανοφάνειας
* Epochs: Επιλέξαμε 100 εποχές κρίνοντας από τα αποτελέσματα της καμπύλης μάθησης.



Στην καμπύλη μάθησης παρατηρούμε την μείωση της ακρίβειας στα δεδομένα εκπαίδευσης που συνάδει με την μείωση του overfitting, ενώ αυξάνεται η ακρίβεια στα Development δεδομένα που δείχνει ότι ο αλγόριθμος μαθαίνει.



AdaBoost:

Για την υλοποίηση του αλγορίθμου AdaBoost χρησιμοποιήσαμε το DecisionTreeClassifier του scikit-learn με βάθος ίσο με 1 σαν weak learner.

Η διεπαφή αποτελείται από τις παρακάτω μεθόδους:

* \_\_init\_\_(amount\_of\_says, models): O κατασκευαστής των αντικειμένων AdaBoost. Παίρνει ως ορίσματα πίνακα models που περιέχει decision stumps και έναν πίνακα με τα amount of say των δέντρων αυτών. Αν δε δοθούν ορίσματα, αρχικοποιούνται ως None.
* get\_params(): Υλοποιούμε για συμβατότητά με μεθόδους του sci-kit learn όπως learning\_curve()
* fit(x\_train\_input, y\_train\_input, M): Παίρνει ως ορίσματα τα δεδομένα εκπαίδευσης και τον αριθμό επαναλήψεων. Αρχικά διαιρούμε τα δεδομένα σε εκπαίδευσης και επικύρωσης 80-20 αντίστοιχα. Έπειτα, αρχικοποιούνται ισάξια βάρη για καθένα από τα παραδείγματα εκπαίδευσης και αρχίζουν οι επαναλήψεις. Σε κάθε επανάληψη, δημιουργείται ένα δέντρο απόφασης βάθους 1 το οποίο εκπαιδεύεται στα δεδομένα και ταξινομεί τα παραδείγματα. Ύστερα, υπολογίζεται το σφάλμα του δέντρου, το amount of say και ενημερώνονται τα βάρη και τα δεδομένα. Αν το σφάλμα είναι μεγαλύτερο από 50%, η επανάληψη επαναλαμβάνεται, αλλιώς τα στοιχεία του δέντρου αποθηκεύονται και συνεχίζουμε με την επόμενη επανάληψη.
* update\_data(x, y): Δέχεται ως ορίσματα τα δεδομένα εκπαίδευσης. Στην αρχή αρχικοποιούμε ένα καινούριο άδειο dataset και δημιουργούμε έναν πίνακα (distribution) ο οποίος σε κάθε i-στή θέση περιέχει το άθροισμα των βαρών μέχρι και το i-στό βάρος. Στη συνέχεια, διαλέγουμε τυχαία έναν αριθμό από το 0 έως το 1 και βρίσκουμε το πρώτο στοιχείο του distribution το οποίο είναι μεγαλύτερο ή ίσο του. Βρίσκουμε σε ποιο παράδειγμα αντιστοιχεί το βάρος αυτό και το προσθέτουμε στην καινούρια βάση. Επαναλαμβάνουμε μέχρι το νέο dataset γεμίσει με δεδομένα. Τέλος, αρχικοποιούμε ξανά ισάξια τα βάρη και επιστρέφουμε τα νέα δεδομένα.
* change\_weights(pred, y, amount\_of\_say): Τα ορίσματα της μεθόδου είναι οι προβλέψεις ενός δέντρου απόφασης, οι πραγματικές τιμές και το amount of say του δέντρου αυτού. Μειώνουμε τα βάρη στα παραδείγματα που το δέντρο αξιολόγησε σωστά και αυξάνουμε τα βάρη στα παραδείγματα που το δέντρο κατάταξε λανθασμένα. Με αυτόν τον τρόπο, στις επόμενες επαναλήψεις δίνεται περισσότερη βαρύτητα στα λανθασμένα δεδομένα. Πριν επιστρέψουμε τα βάρη, φροντίζουμε το άθροισμα τους να είναι ίσο με τη μονάδα.
* predict(x): Δέχεται ως είσοδο ένα σύνολο παραδειγμάτων. Για κάθε δέντρο που δημιουργήθηκε, υπολογίζουμε το γινόμενο του amount of say με την απόφαση του δέντρου για κάθε παράδειγμα. Οι τελικές αποφάσεις είναι ό,τι αποφάσισε η πλειοψηφία των δέντρων για κάθε παράδειγμα.
* evaluate(y\_true, y\_predicted): Οι παράμετροι είναι οι πραγματικές κατηγορίες κάποιων παραδειγμάτων και οι κατηγορίες στις οποίες τα κατάταξε ο αλγόριθμος μας. Υπολογίζει τις τιμές των precision, recall, f1 χρησιμοποιώντας τις μεθόδους του scikit learn.
* classification\_diagrams(x\_train, y\_train, x\_test, y\_test, train\_sizes): Παράγει τρία plots για την πορεία των precision, recall, f1 score συναρτήσει του μεγέθους των δεδομένων εκπαίδευσης.
* custom\_classification\_report(*x\_train*, *y\_train*,*x\_test*, *y\_test*, *train\_sizes*): Καλεί την classification\_diagrams() και την my\_learning\_curve() για να δώσει μια πλήρη εικόνα των αποτελεσμάτων μάθησης του ταξινομητή.

Αποτελέσματα:

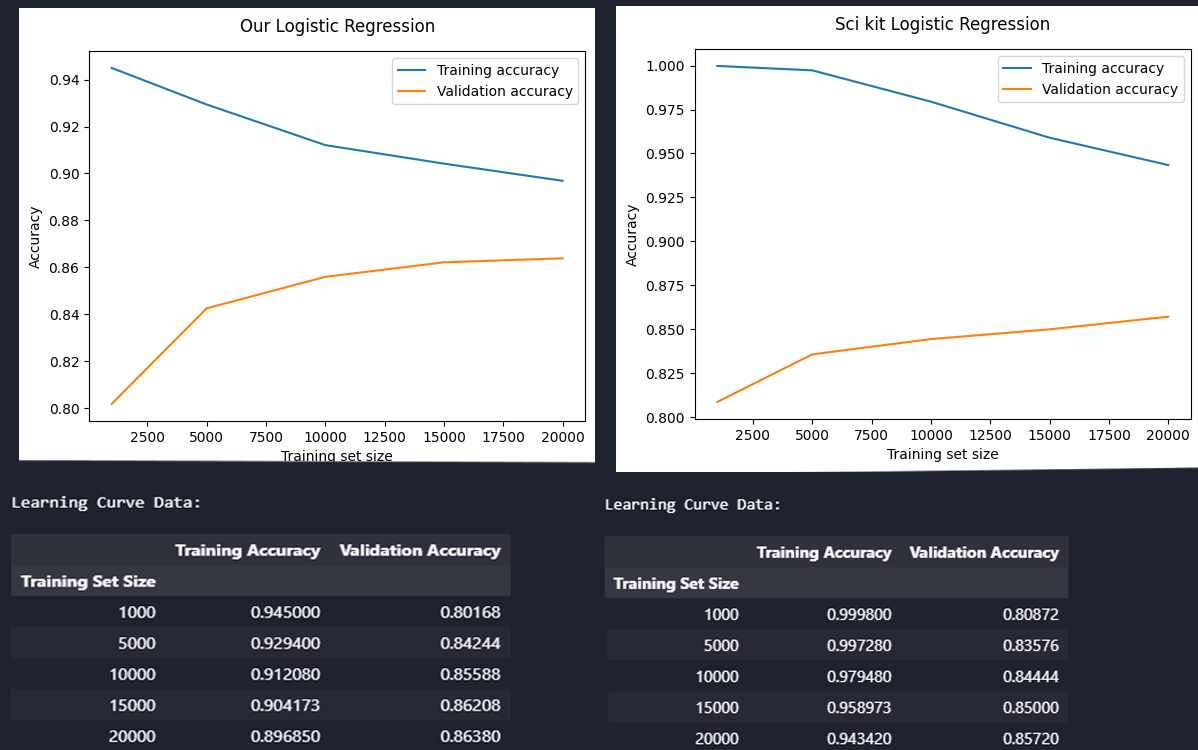
2ο Μέρος

Σε αυτό το σημείο θα συγκρίνουμε τις υλοποιήσεις των αλγορίθμων μας με τις υλοποιήσεις του sci-kit learn. Για αυτή την σύγκριση πέρα τον μετρικών ακρίβειας όπως accuracy, precision, recall και f1 score έχουμε υλοποιήσει διάφορα διαγράμματα όπως precision-recall καμπύλη, ROC καμπύλη και καμπύλη μάθησης.

Logistic regression:

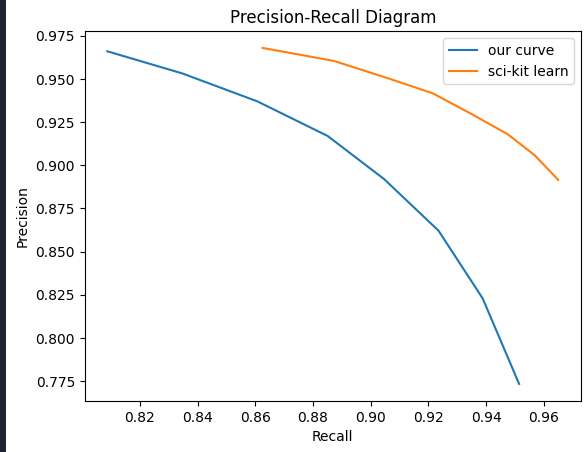
Καμπύλες μάθησης:

Παρατηρούμε παρόμοια πορεία στις δυο καμπύλες μάθησης με την δικιά μας υλοποίηση να φτάνει σε μικρότερη ακρίβεια στα δεδομένα εκπαίδευσης που ίσως σημαίνει λιγότερο overfitting ενώ πετυχαίνει σχεδόν ίση ακρίβεια στα δεδομενα επικύρωσης.



Precision-Recall διάγραμμα:

Αλλάζοντας το threshold σε διάφορες τιμές βλέπουμε πως αλλάζει η ισορροπία precision-recall και παρατηρούμε ότι η υλοποίηση παρουσιάζει ελαφρώς καλυτέρα αποτελέσματα για τις διάφορες τιμές threshold.



Καμπύλες ROC:

Με τις καμπύλες roc βλέπουμε την ικανότητα ενός ταξινομητή να ξεχωρίσει θετικές από αρνητικές κριτικές. Από το AUC (area under curve) παρατηρούμε ότι η υλοποίηση μας έχει ελαφρός καλύτερη απόδοση από αυτή του scikit learn με τις παραμέτρους που έχουν χρησιμοποιηθεί.

